LCC 6310
The Computer as an Expressive Medium

Lecture 5

Overview

Programming concepts
- Methods
- Classes
  - *Processing*, pp.301-318

Drawing a rocket

```
background(0);
fill(255);
triangle(10, 0, 0, 20, 20, 20);
rectMode(CORNERS);
rect(5, 20, 8, 23);
rect(12, 20, 15, 23);
```

Let's try this in Processing...

Now I want to draw several rockets

I want several rockets in different locations on the screen

I could copy and paste the code
- But I would need to adjust all the numbers for the new locations

Or... I can just define a method...

First method for drawing a rocket

```
void drawRocket() {
  fill(255);
  triangle(10, 0, 0, 20, 20, 20);
  rectMode(CORNERS);
  rect(5, 20, 8, 23);
  rect(12, 20, 15, 23);
}
```

Note: we can call the method whatever we want, but it's conventional to give methods names that begin with a lower case letter E.g. drawRocket rather than DrawRocket

Now how do we use this code? Can we just call the method at the top of the file? Let's try it...

First method for drawing a rocket

```
void drawRocket() {
  fill(255);
  triangle(10, 0, 0, 20, 20, 20);
  rectMode(CORNERS);
  rect(5, 20, 8, 23);
  rect(12, 20, 15, 23);
}
```


Now how do we use this code? Can we just call the method at the top of the file? Let's try it... causes an error!!

You can't just directly call drawRocket() at the top of the file

Once you start using methods, all code must be in methods...

Calling drawRocket()

```
void setup() {
  drawRocket();
}

void drawRocket() {
  fill(255);
  triangle(10, 0, 0, 20, 20, 20);
  rectMode(CORNERS);
  rect(5, 20, 8, 23);
  rect(12, 20, 15, 23);
}
```

This didn't seem to help much…

Our method still just draws a rocket at one fixed location

We need arguments that allow us to tell the program where we want the rocket!

- This means we need to figure out the relationship between the position of the rocket and the location of the rest of its parts

Note: Argument variables are available within the method, but not outside (method scope)

drawRocket() with arguments

```
void drawRocket(int noseX, int noseY) {  // draw a rocket with respect to its nose
  // other parts of the rocket with respect to its nose
  int bottomOfRocket = noseY + 20;
  int leftOfRocket   = noseX - 10;
  int rightOfRocket  = noseX + 10;

  fill(255);

  // draw the rocket body
  triangle(noseX, noseY, leftOfRocket, bottomOfRocket, rightOfRocket, bottomOfRocket);

  // draw the rocket feet
  rectMode(CORNERS);
  rect(leftOfRocket + 5, bottomOfRocket, leftOfRocket + 8, bottomOfRocket + 3);
  rect(rightOfRocket - 8, bottomOfRocket, rightOfRocket - 5, bottomOfRocket + 3);
}
```

Now to specify rotation

Currently we specify the nose position and it draws a rocket that faces straight up

To rotate the rocket, we would need to figure out the new position of the vertices

- That's hard!

Instead of rotating the rocket, let's rotate the paper

- As long as we're rotating the paper, might as well slide it around as well

To prepare for translating and rotating, first let's draw the rocket centered around the origin (0,0)

Centering the rocket around the origin

Currently the rocket draws with respect to its nose

- You can see this by calling drawRocket(0,0)

Now we want to draw the rocket around the origin

We know the rocket dimensions are:

```
int bottomOfRocket = noseY + 20;
int leftOfRocket   = noseX - 10;
int rightOfRocket  = noseX + 10;
```

From this, we know our rocket has a width and height of 20 pixels, so we can define some constants:

```
final int halfHeight = 10;
final int halfWidth = 10;
```

Now let's use these to draw the rocket…

Body and feet around the origin

Drawing the rocket body used to look like:

```
triangle(noseX, noseY, leftOfRocket, bottomOfRocket, rightOfRocket, bottomOfRocket);
```

Now instead we start at the origin and use halfHeight and halfWidth:

```
triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);
```

Drawing the rocket feet used to look like:

```
rect(leftOfRocket + 5, bottomOfRocket, leftOfRocket + 8, bottomOfRocket + 3);
rect(rightOfRocket - 8, bottomOfRocket, rightOfRocket - 5, bottomOfRocket + 3);
```

Now instead we center around the origin:

```
rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);
```

Putting it together

```
void drawRocket(int x, int y, float rot) {
  final int halfHeight = 10;
  final int halfWidth = 10;

  triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);
  rectMode(CORNERS);
  rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
  rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);
}
```

Note: we're purposely ignoring the arguments for now!

Now add translation and rotation

```
void drawRocket(int x, int y, float rot) {
  final int halfHeight = 10;
  final int halfWidth = 10;

  translate(x, y);
  rotate(rot);

  triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);
  rectMode(CORNERS);
  rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
  rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);
}
```

Let's try drawing several rockets…

```
drawRocket(50,50,HALF_PI);
drawRocket(25,25,HALF_PI);
```

Works fine for one call…

- But when we call it twice, it doesn't seem to work
  - Looks like first call messes up subsequent calls
- What is happening?!
  - When you move the paper (translate, rotate) the paper stays moved
  - Subsequent paper moves (additional translates and rotates) build on top of the previous ones
- So what we want is to move the paper, draw, and then move it back
- We need pushMatrix() and popMatrix()
  - pushMatrix() remembers the previous paper position (i.e. the previous transformation matrix)
  - popMatrix() restores the paper to the remembered position (i.e. restores the transformation matrix)

Using pushMatrix() and popMatrix()

```
void drawRocket(int x, int y, float rot) {
  final int halfHeight = 10;
  final int halfWidth = 10;

  pushMatrix();

  translate(x, y);
  rotate(rot);

  triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);
  rectMode(CORNERS);
  rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
  rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);

  popMatrix();
}
```

Let's try drawing several rockets now…

```
drawRocket(50,50,HALF_PI);
drawRocket(25,25,HALF_PI);
```

Classes

Java (Processing) is an object-oriented language

This means that parts of your program that you treat as conceptual things, become things (objects) in the program code

Objects are built from classes

- Classes are the blueprint, objects are built from the blueprint
- Objects are called instances of classes

Our rocket sure seems like a thing – let's turn it into a class

Parts of a class

Classes define fields, constructors and methods

Fields are the variables that appear inside every instance of the class, you can think of them as properties that each object will have

- Each instance has its own values
- E.g. a square class might have variables for size and color, and each instance of a square can have its own size and color

Constructors are special methods that define how to build instances (generally, how to set the initial values of fields)

- A class can have multiple constructors that can be used to set up the object in different ways

Methods are how you do *things* to instances, you can think of them as the operations that can happen to an object

- E.g. a square can be drawn, its color can be changed, etc.

Defining the Rocket class

```
class Rocket {
  // fields
  float rotation = 0;
  float xPos;
  float yPos;
  final int halfWidth = 10;
  final int halfHeight= 10;

  // constructor
  Rocket(int initialX, int initialY, float initialRot) {
    xPos = initialX;
    yPos = initialY;
    rotation = initialRot;
  }
}
```

Using the class to create instances

Classes define a type

You can now declare variables of this type and initialize them using the constructor

Like arrays, the keyword new is used to tell Java to create a new object (hmm, so arrays must be objects…)

```
void setup() {
  Rocket r1 = new Rocket(75, 10, 0);
  Rocket r2 = new Rocket(50, 50, PI/2);
}
```

Let's see try this…

Nice, but my rockets don't do anything

- Oh yeah, you need methods to do things to objects…

Adding a draw routine to our Rocket

```
void drawMe() {
  pushMatrix();
  translate(xPos, yPos);
  rotate(rotation);

  triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);

  rectMode(CORNERS);
  rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
  rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);
  popMatrix();
}
```

- We don't need arguments because we use the fields
  - But we could define additional arguments if we wanted to

Calling methods on objects

- You call methods on instances
  - Think of the method as something your asking the object to do
- For example, we can now ask the rockets to draw themselves
  - `r1.drawMe();`
- In general, to call a method, take the name of the variable holding the object + "." + the method name
  - `myObject.myMethod();`
- Let's try drawing our rockets…

Adding a color argument

```
void drawMe(color col) {
  pushMatrix();
  translate(xPos, yPos);
  rotate(rotation);

  fill(col);

  triangle(0, -halfHeight, -halfWidth, halfHeight, halfWidth, halfHeight);
  rectMode(CORNERS);
  rect(-halfWidth + 5, halfHeight, -halfWidth + 8, halfHeight + 3);
  rect(halfWidth - 8, halfHeight, halfWidth - 5, halfHeight + 3);
  popMatrix();
}
```

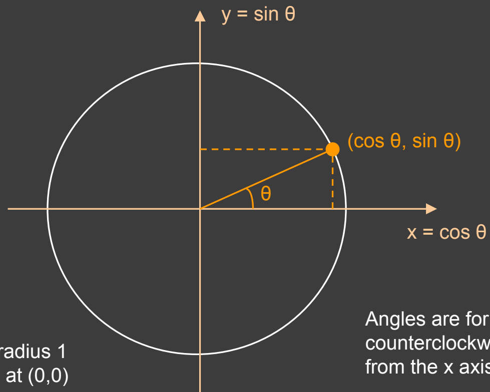
- Now we can draw our rockets with color:

```
  r1.drawMe(125);
  r2.drawMe(color(255,125,0));
```

What else could we do to the Rocket?

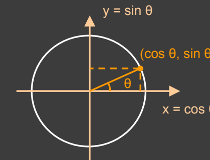
- We may want to change the rotation
  - rotateClockwise();
  - rotateCounterclockwise();
- We may want to give the rocket a boost
  - fireThrusters();
- Let's see a full implementation of our Rocket class…
  - Do you understand how the rocket moves?

Understanding the rocket movement: angles, velocity and thrust



circle of radius 1 centered at (0,0)

Angles are formed counterclockwise from the x axis



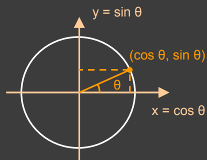
Velocity components

Suppose a rocket is moving at an angle of $\theta = \pi/6$ (i.e. 30 degrees) like in the picture. We can think of this in terms of x and y as follows:

- The rocket it is moving in both the x and y directions. So it has both x and y velocity.
- The rocket is moving more in the x direction than in the y direction. So the x velocity is greater than the y velocity.

Suppose the rocket velocity is known, with components velocityX and velocityY. Then we can update the position of the rocket (xPos, yPos) based on the time since the last update as follows:

```
xPos = xPos + velocityX * timeSinceLastDraw;
yPos = yPos + velocityY * timeSinceLastDraw;
```



fireThrusters: an extra push

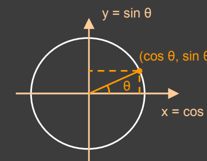
Now suppose we want to give the rocket and extra little push, so that it keeps moving in the same direction but a little faster.

- We need to increase velocityX and velocityY by some number each, call these thrustX and thrustY. What will these numbers be?

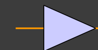
In our $\theta = \pi/6$ example, we can guess that the number added to velocityX will be bigger than the number added to velocityY, since the rocket is moving more in the x direction than in the y direction.

- The question is: how much bigger? How can we exactly determine the x and y velocity components?

The diagram above gives the answer. Let's look at some examples…



Example 1

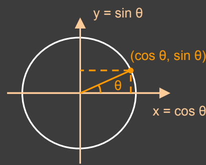


If the rocket is moving in the x direction only (i.e. $\theta = 0$), what does the extra thrust look like in terms of x and y directions?

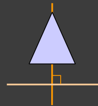
All thrust is in the x direction, no thrust in the y direction.

That means:
xThrust = cos (θ) = cos (0) = 1
yThrust = sin (θ) = sin (0) = 0

And we can update our velocities as follows:
velocityX = velocityX + xThrust = velocityX + 1
velocityY = velocityY + yThrust = velocityY



Example 2

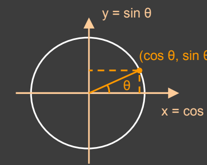


If the rocket is moving in the y direction only (i.e. $\theta = \pi/2$), what does the extra thrust look like in terms of x and y directions?

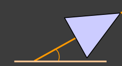
All thrust is in the y direction, no thrust in the x direction.

That means:
xThrust = cos (θ) = cos (π/2) = 0
yThrust = sin (θ) = sin (π/2) = 1

And we can update our velocities as follows:
velocityX = velocityX + xThrust = velocityX
velocityY = velocityY + yThrust = velocityY + 1



Example 3



If the rocket is moving at an angle $\theta = \pi/6$ (30 degrees), what does the extra thrust look like in terms of x and y directions?

Greater thrust in the x direction than in the y direction.

That means:
xThrust = cos (θ) = cos (π/6) = 0.8660
yThrust = sin (θ) = sin (π/6) = 0.5

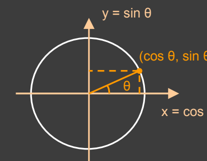
And we can update our velocities as follows:
velocityX = velocityX + xThrust = velocityX + 0.8660
velocityY = velocityY + yThrust = velocityY + 0.5

Acceleration

Suppose we want the rocket to speed up even more when we fire the thrusters…

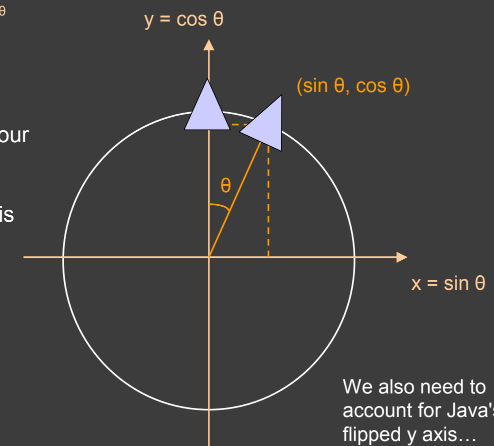
- We can simply multiply the xThrust and yThrust components by a chosen acceleration factor:
  - velocityX = velocityX + xThrust * a
  - velocityY = velocityY + yThrust * a
- Or using our rocket's angle, this is the same as saying:
  - velocityX = velocityX + cos (θ) * a
  - velocityY = velocityY + sin (θ) * a

Of course when translating these concepts into code, we need to account for Java's coordinate system…



Java and rocket rotation

Note that in Java, our rocket rotates clockwise from an angle of 0 when it is facing upwards



We also need to account for Java's flipped y axis…

Java coordinates

y = sin θ
(cos θ, sin θ)
θ
x = cos θ

(sin θ, - cos θ)
θ
x
y

In Java, the y coordinates increase from top to bottom. This means we need to put a negative (-) sign in front of the y coordinate of the rocket, i.e. (sin θ, - cos θ)

Putting it all together…

We can now translate our velocity equations for the fireThrusters() method into code as follows:

```
// thrust in x direction, acceleration is 1
velocityX = velocityX + sin(rotation) * 1;

// thurst in x direction, acceleration is 1
velocityY = velocityY - cos(rotation) * 1;
```

A hack to fix the focus problem

```
void setup() {
  background(0);
  size(400, 400);
  requestFocus();
}
...
void focusLost(FocusEvent e) {
  requestFocus();
}
```

Let's see this in Processing…

Remember…

For Thursday this week: Theory Readings

- Two presenters (you know who you are!)
- Everyone else: prepare one discussion question for each reading
  - *Sketchpad: A Man-Machine Graphical Communication Systems* - Sutherland (NMR p.109)
  - *Direct Manipulation: A Step Beyond Programming Languages* - Schneiderman (NMR p.485)